USF UNIVERSITY OF SAN FRANCISCO

CHANGE THE WORLD FROM HERE

Volatile Keyword

Chris Bryan ◆ cbryan2@usfca.edu
Department of Computer Science

synchronized keyword

- Protects blocks of code, **not** objects
- Provides **mutual exclusion**, which causes blocking, which slows down code
- Can be used to prevent thread interference (**atomicity**) and memory consistency errors (**visibility**) of shared resources

http://docs.oracle.com/javase/tutorial/essential/concurrency/sync.html

volatile keyword

- Indicates a variable is **unstable**, and may be accessed concurrently
- Think of it as “synchronized lite”
  - Less code, less runtime overhead, but only a subset of functionality compared to synchronized!
  - Changes are always **visible** to other threads
  - Does not causes blocking
  - ☹ You don’t get **atomicity**
- Does not eliminate need for other synchronization!

http://docs.oracle.com/javase/tutorial/essential/concurrency/atomic.html

volatile keyword

- Threads always read latest value (not cached value)
- Write operations cannot depend on current value
  - e.g. shutdown = true;
- Read operations cannot be used with other variables
  - e.g. ~~if (volatileVar < otherVar)~~
  - e.g. if (volatileVar == true)

http://www.ibm.com/developerworks/java/library/j-jtp06197/index.html

Proper Use Patterns

Java Theory and Practice: Managing Volatility
IBM developerWorks
http://www.ibm.com/developerworks/java/library/j-jtp06197/index.html

Proper Use Patterns

- **Pattern #1: Status flags**
  - Write of flag does not depend on current value
  - Read of flag does not depend on other variables
- **Pattern #2: One-Time Safe Publication**
  - Object must be thread-safe or effectively immutable
  - Object must be initialized only once

http://www.ibm.com/developerworks/java/library/j-jtp06197/index.html

```
1  private volatile boolean active;
2
3  public void shutdown() {
4      active = false;
5  }
6
7  public void run() {
8      while (active) {
9          // do stuff...
10     }
11 }
```

Pattern #1: Status Flag, http://www.ibm.com/developerworks/java/library/j-jtp06197/index.html

```
1  public class WidgetLoader extends Thread {
2      public volatile Widget widget;
3      public void run() {
4          widget = loadWidget();
5      }
6  }
7
8  public class MainThread extends Thread {
9      public void run() {
10         while (true) {
11             if (widgetLoader.widget != null) {
12                 // do stuff...
```

Pattern #2: One-Time Safe Publication, http://www.ibm.com/developerworks/java/library/j-jtp06197/index.html

Proper Use Patterns

- **Pattern #3: Independent Observations**
  - Similar to one-time safe publication, except multiple independent writes of effectively immutable object
- **Pattern #5: "Cheap" Read-Write Lock**
  - Use volatile for non-blocking reads
  - Use synchronized for blocking writes

http://www.ibm.com/developerworks/java/library/j-jtp06197/index.html

```
1  private volatile String lastUser;
2
3  public void auth(String user, String pass) {
4      boolean valid = checkPass(user, pass);
5      if (valid) {
6          activeUsers.add(user);
7          lastUser = user;
8      }
9      return valid;
10   }
11 }
```

Pattern #3: Independent Observations, http://www.ibm.com/developerworks/java/library/j-jtp06197/index.html

```
1  private volatile int counter;
2
3  public int getCount() {
4      return counter;
5  }
6
7  public synchronized void increment() {
8      counter++;
9  }
```

Pattern #5: Cheap Read/Write Lock, http://www.ibm.com/developerworks/java/library/j-jtp06197/index.html

Conclusion

- Use **carefully**, or not at all
  - None of the code for this class **requires** the use of this keyword
- Use for **simplicity** when full synchronization is not necessary
- Use for **scalability** when reads outnumber writes
  - Or, use an actual read/write lock

http://www.ibm.com/developerworks/java/library/j-jtp06197/index.html

Chris Bryan ◆ cbryan2@usfca.edu
Department of Computer Science